

IA et Nous

Réseaux de neurones

*Application à reconnaissance
chiffres manuscrits*

La reconnaissance de chiffres manuscrits

- 1. Modèle simple à 10 neurones*
- 2. Modèle à 5 couches de neurones*
- 3. Limites*
- 4. Vers la Convolution*

Sources :

Yann Lecun

Devon France : Martin Gorner <https://www.youtube.com/watch?v=BtAVBeLuigI>



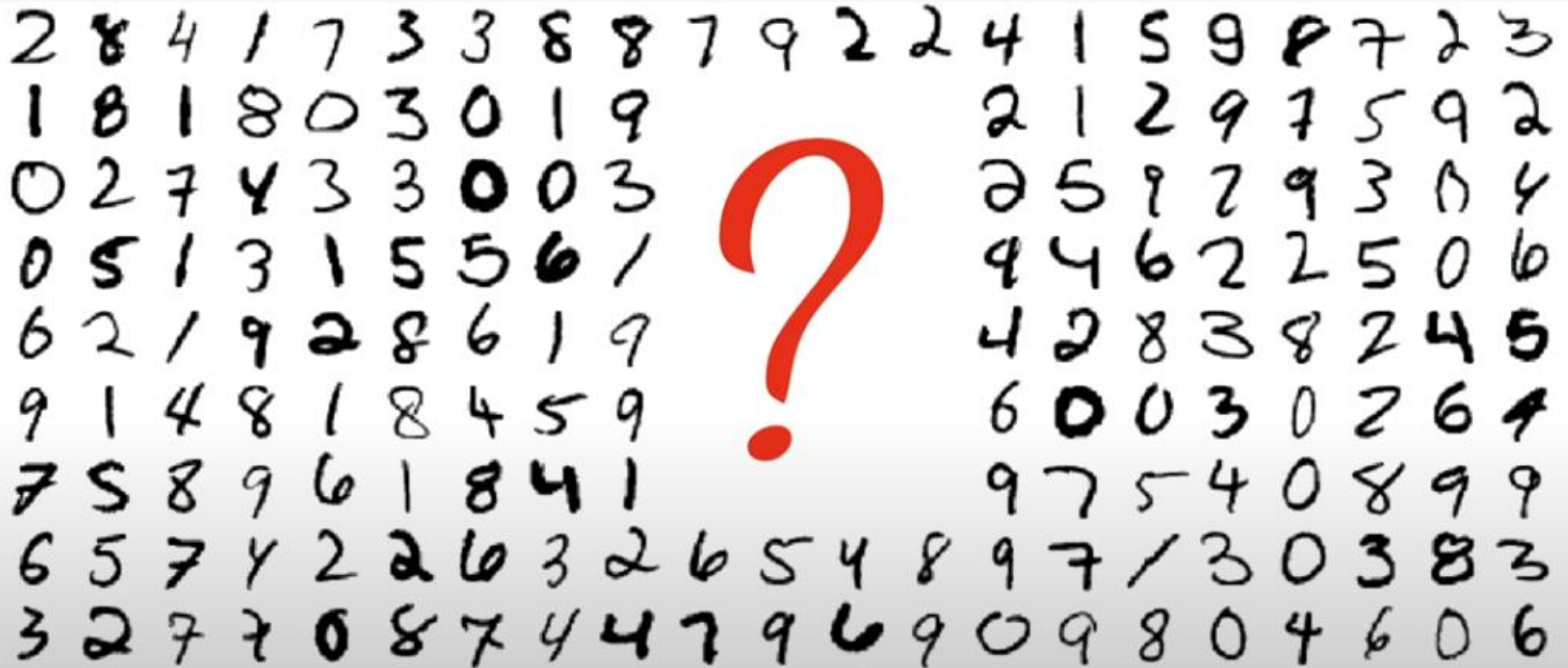
Martin Görner

Google Developer relations

[@martin_gorner](#)

plus.google.com/+MartinGorner

Reconnaissance chiffres manuscrits



MNIST = Mixed National Institute of Standards and Technology - Download the dataset at <http://yann.lecun.com/exdb/mnist/>

Apprentissage automatique

*Exemple classique : les chiffres manuscrits *.*

Comment les reconnaître?

Dans les années 80 => Systèmes Experts

Règles :

- *Si les pixels noirs ont la forme d'une seule boucle, alors c'est un 0*
- *si deux boucles entrelacées alors c'est un 8*
- *si ligne verticale au milieu , alors c'est un 1*

Limites vite atteintes :

- *règles pas suffisamment précises, nécessité de définir ce qu'est une boucle, le milieu, etc...*
- *nécessité d'une infinité d'exceptions à ces règles selon l'écriture cf. exemple .*

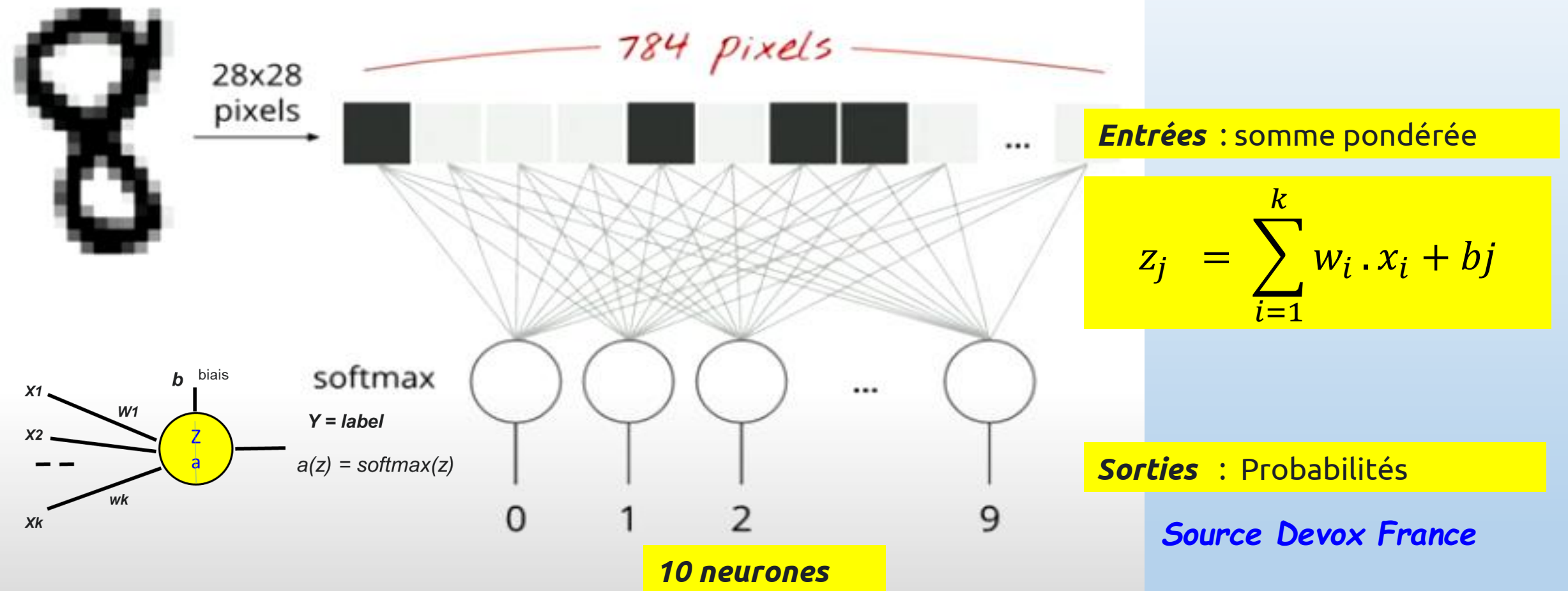
➔ Autre méthode avec le connexionnisme

* Exemple tiré de la base de données MNIST Modified «National Institute of Standards and Technology».

Modèle simple Classification Softmax

Une couche de 10 neurones

Apprentissage supervisé



Fonction softmax

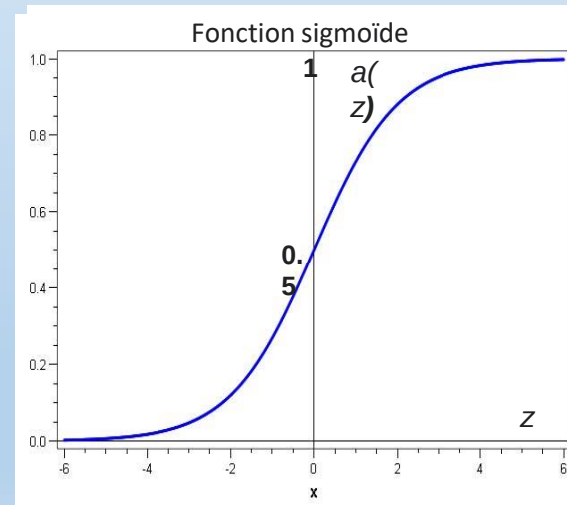
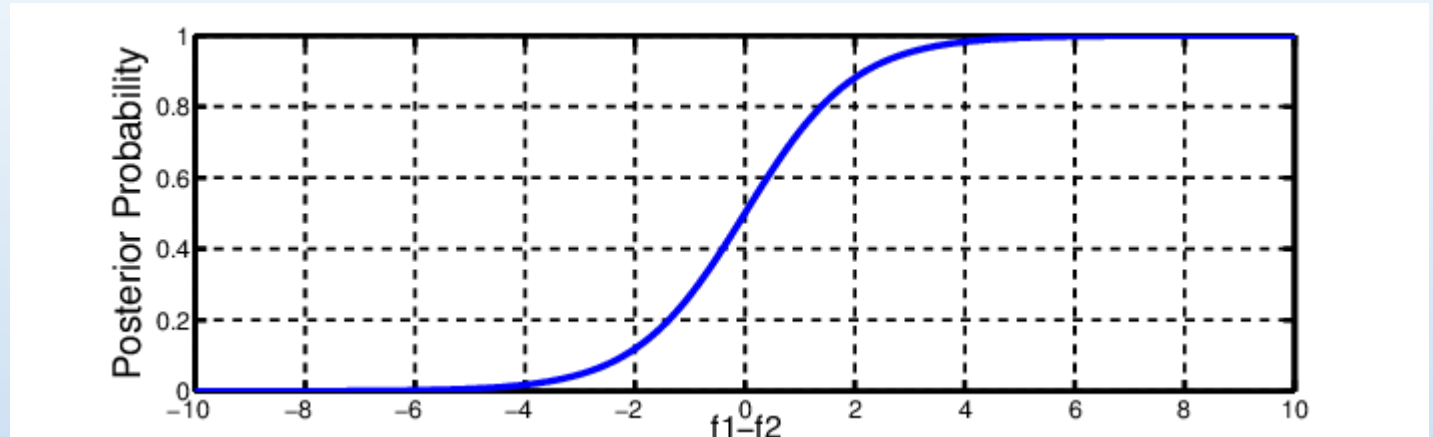
Ou fonction exponentielle normalisée

Le softmax opère sur un vecteur

$$\text{Softmax } \sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

$$\text{Sigmoid } S(x) = \frac{1}{1 + e^{-x}}$$

Le sigmoïde opère sur un scalaire.



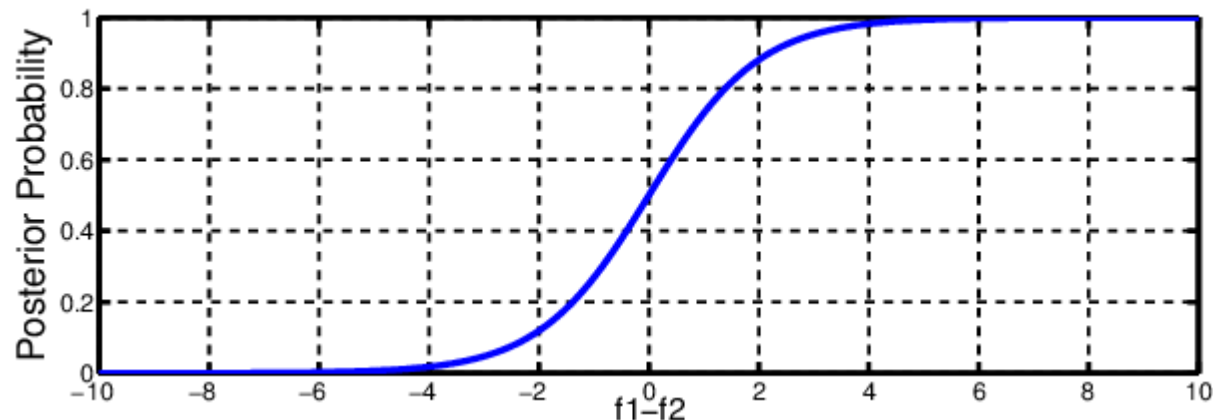
Fonction softmax

Le softmax opère sur un vecteur

$$\text{Softmax } \sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Entrées : vecteur de k nombres réels

Sorties : vecteur de k nombres réels
>0, et dont la somme est = 1
=> **Probabilités**



Considérons un vecteur

$$\mathbf{z} = (1 ; 3 ; 2,5 ; 5 ; 4 ; 2)$$

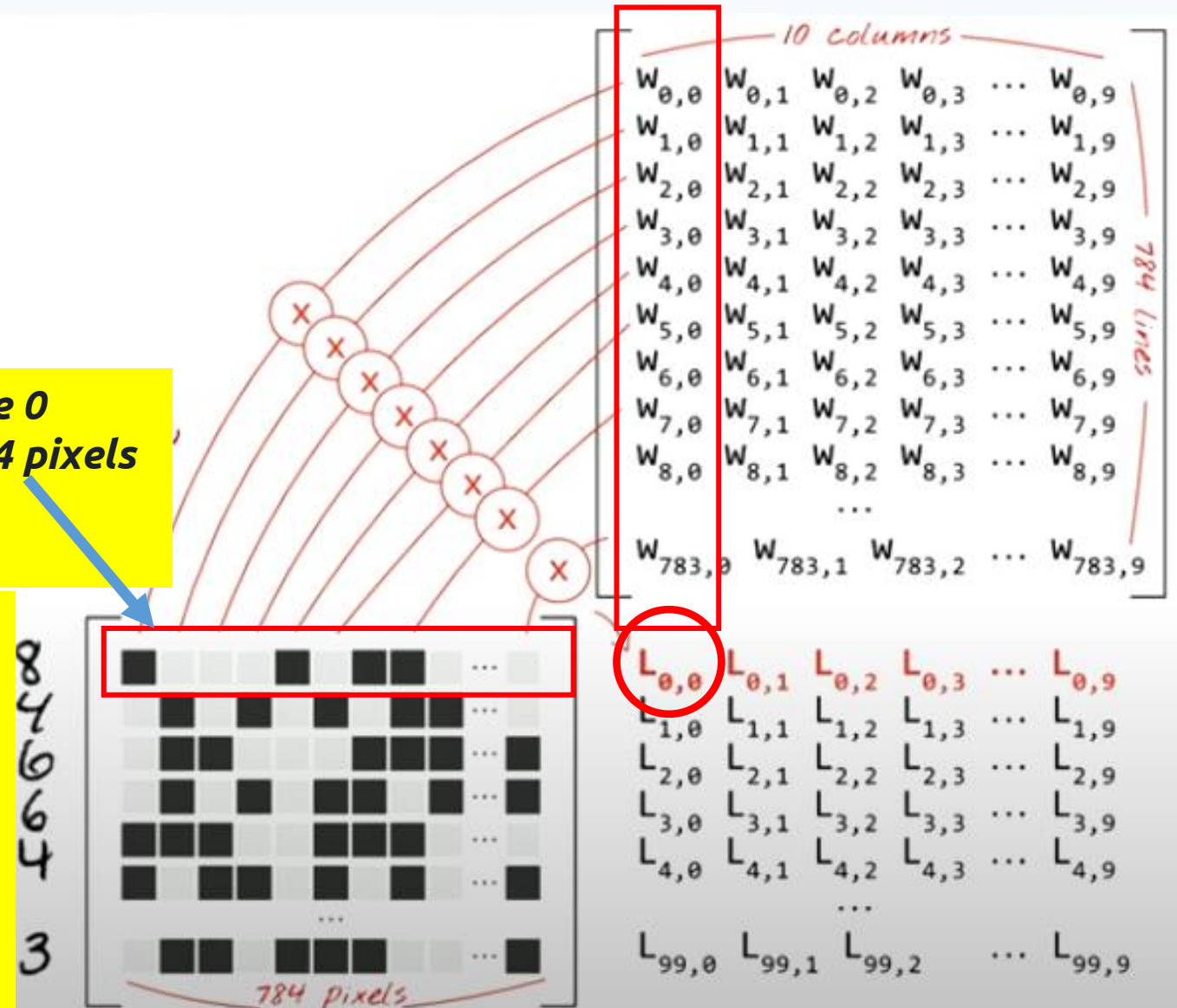
de six nombres réels. La fonction softmax donne en sortie (tronquée à 10^{-2}) :

$$\sigma(\mathbf{z}) = (0,01; 0,08; 0,04; 0,60; 0,22; 0,03).$$

Avec 100 images

image 0
de 784 pixels

100
images



1 colonne = 1 neurone

$$L = X.W + b$$

Source Devox France

Softmax

$$Y = \text{softmax}(L)$$

$$L = X \cdot W + b$$

tensor shapes: $X[100, 784]$ $W[784, 10]$ $b[10]$

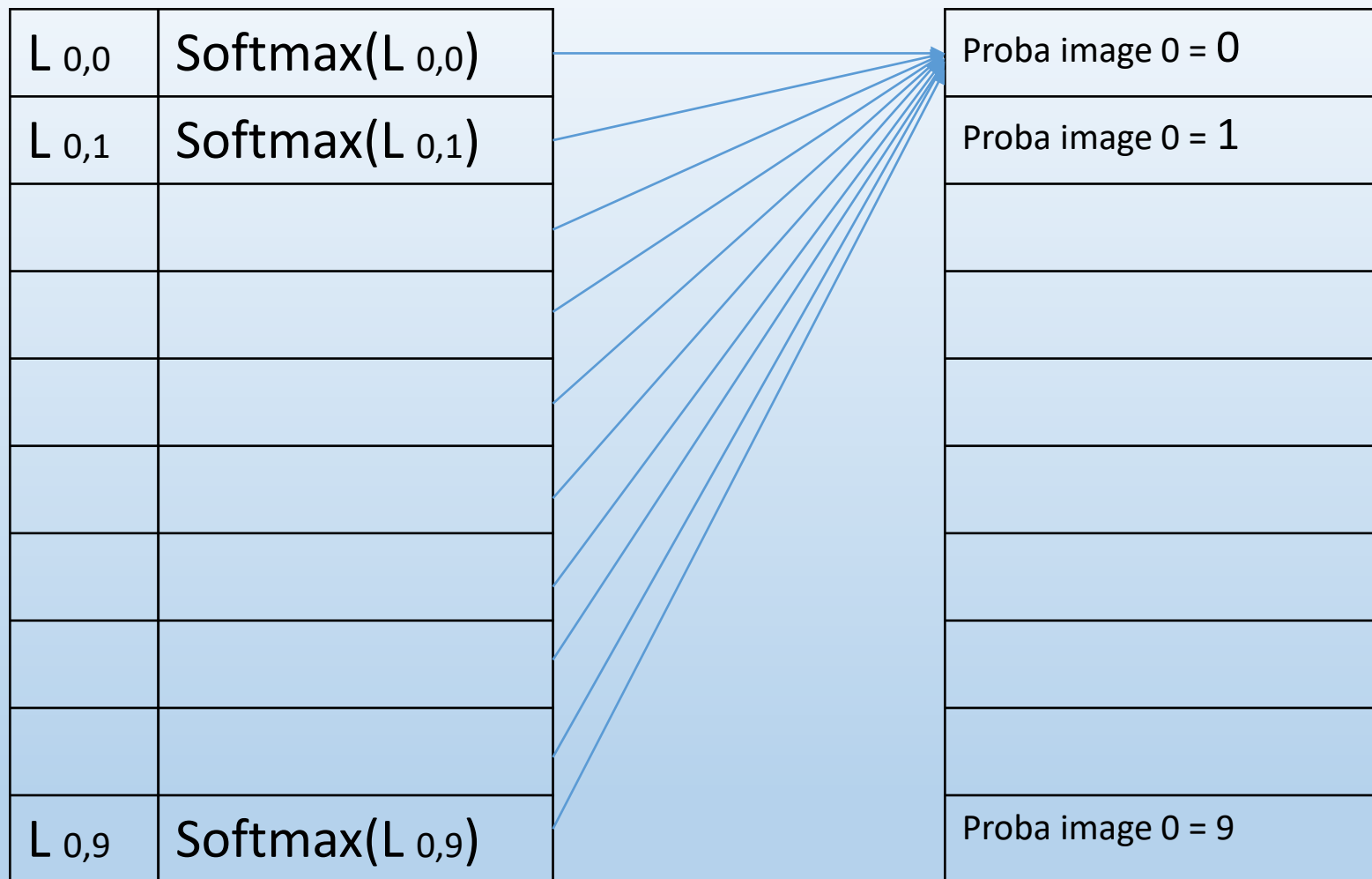
```
Y = tf.nn.softmax(tf.matmul(X, W) + b)
```

matrix multiply

broadcast on all lines

Softmax

$$L = X \cdot W + b$$



Fonction Soft max

Illustration

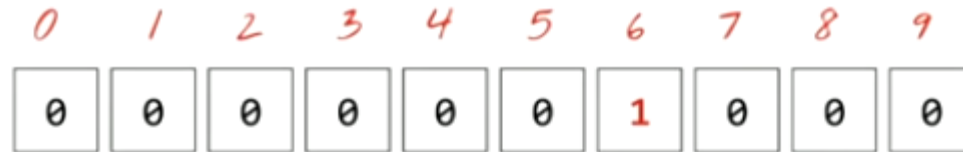
Explicateur CNN

<https://poloclub.github.io/cnn-explainer/>

Entraînement

Source Devox France

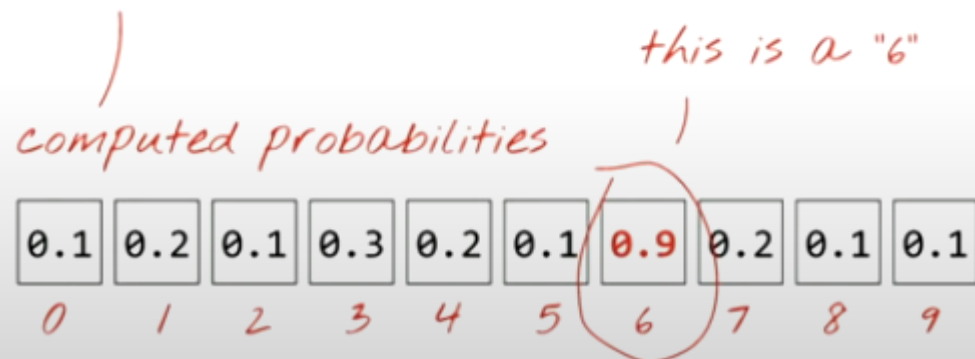
Descente de gradient
Minimisation de la fonction
de coût



actual probabilities, "one-hot" encoded

Fonction de coût utilisée

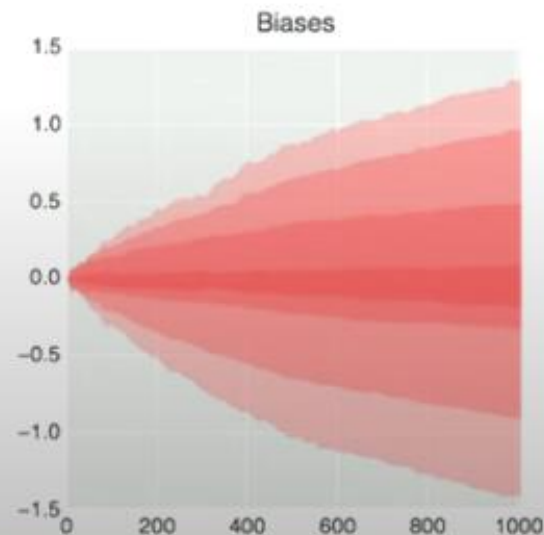
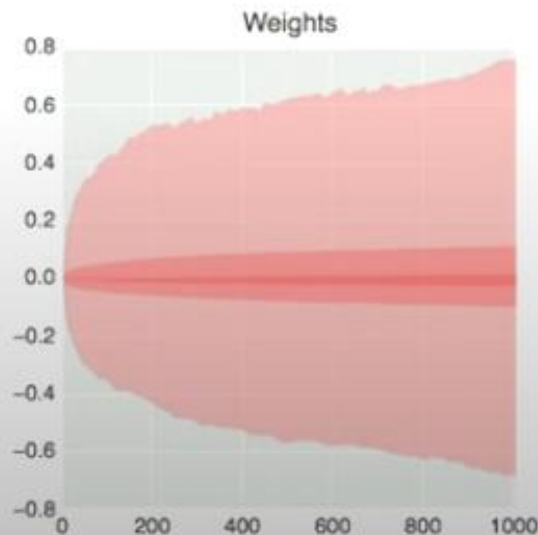
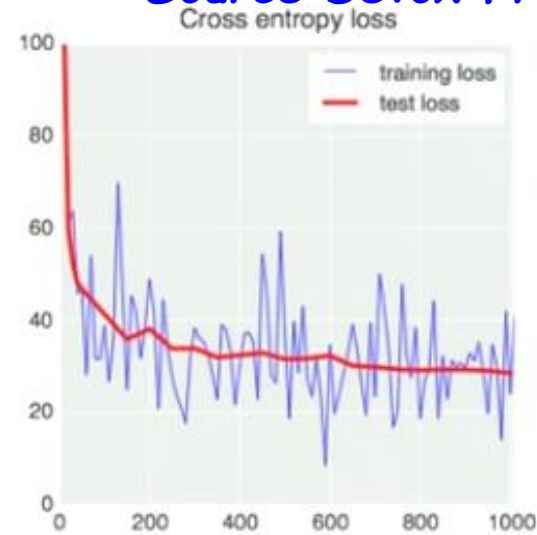
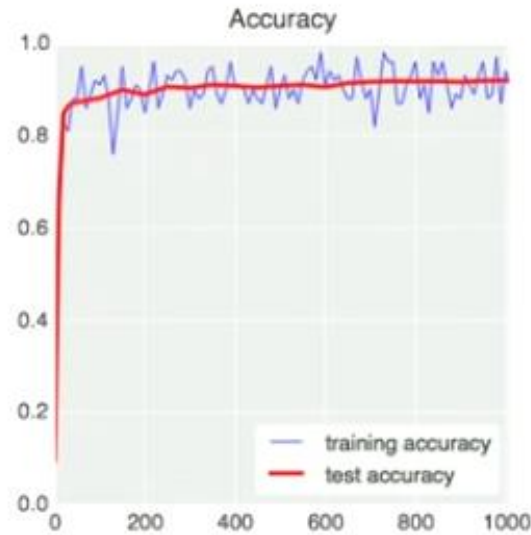
Cross entropy: $-\sum Y_i' \cdot \log(Y_i)$



computed probabilities

Entrainement

Source Devox France



Entrainement

Lots de 100 images

**(60 000 chiffres
d'entrainement)**

Test

**10 000 images non
connues**

Perfo du modèle 92 % !!!!

```
import tensorflow as tf
```

this will become the batch size, 100

```
X = tf.placeholder(tf.float32, [None, 28, 28, 1])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
```

28 x 28 grayscale images

```
init = tf.initialize_all_variables()
```

Training = computing variables W and b

Source Devox France

```
# model
Y = tf.nn.softmax(tf.matmul(tf.reshape(X, [-1, 784]), W) + b)
# placeholder for correct answers
Y_ = tf.placeholder(tf.float32, [None, 10])
```

flattening images

"one-hot" encoding

```
# Loss function
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))

# % of correct answers found in batch
is_correct = tf.equal(tf.argmax(Y, 1), tf.argmax(Y_, 1))
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

"one-hot" decoding

```
optimizer = tf.train.GradientDescentOptimizer(0.003)
train_step = optimizer.minimize(cross_entropy)
```

Taux d'apprentissage

Exécution

```
sess = tf.Session()
sess.run(init)

for i in range(1000):
    # Load batch of images and correct answers
    batch_X, batch_Y = mnist.train.next_batch(100)
    train_data={X: batch_X, Y_: batch_Y}

    # train
    sess.run(train_step, feed_dict=train_data)

    # success ?
    a,c = sess.run([accuracy, cross_entropy], feed_dict=train_data)

    # success on test data ?
    test_data={X: mnist.test.images, Y_: mnist.test.labels}
    a,c = sess.run([accuracy, cross_entropy], feed_dict=test_data)
```

*running a Tensorflow
computation, feeding
placeholders*

*Tip:
do this
every 100
iterations*

Affichage Perfo, a et c

```
import tensorflow as tf
```

initialisation

```
X = tf.placeholder(tf.float32, [None, 28, 28, 1])
```

```
W = tf.Variable(tf.zeros([784, 10]))
```

```
b = tf.Variable(tf.zeros([10]))
```

```
init = tf.initialize_all_variables()
```

model

```
Y = tf.nn.softmax(tf.matmul(tf.reshape(X, [-1, 784]), W) + b)
```

placeholder for correct answers

```
Y_ = tf.placeholder(tf.float32, [None, 10])
```

loss function

```
cross_entropy = -tf.reduce_sum(Y_ * tf.log(Y))
```

% of correct answers found in batch

```
is_correct = tf.equal(tf.argmax(Y, 1), tf.argmax(Y_, 1))
```

```
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
```

success metrics

training step

```
optimizer = tf.train.GradientDescentOptimizer(0.003)
```

```
train_step = optimizer.minimize(cross_entropy)
```

```
sess = tf.Session()
```

```
sess.run(init)
```

```
for i in range(10000):
```

Load batch of images and correct answers

```
batch_X, batch_Y = mnist.train.next_batch(100)
```

```
train_data = {X: batch_X, Y_: batch_Y}
```

train

```
sess.run(train_step, feed_dict=train_data)
```

success ? add code to print it

```
a, c = sess.run([accuracy, cross_entropy], feed=train_data)
```

success on test data ?

```
test_data = {X: mnist.test.images, Y_: mnist.test.labels}
```

```
a, c = sess.run([accuracy, cross_entropy], feed=test_data)
```

Run

Deep Learning

Modèle à 5 couches

Sources :

Yann Lecun

Devon France : Martin Gorner <https://www.youtube.com/watch?v=BtAVBeLuigI>



Martin Görner

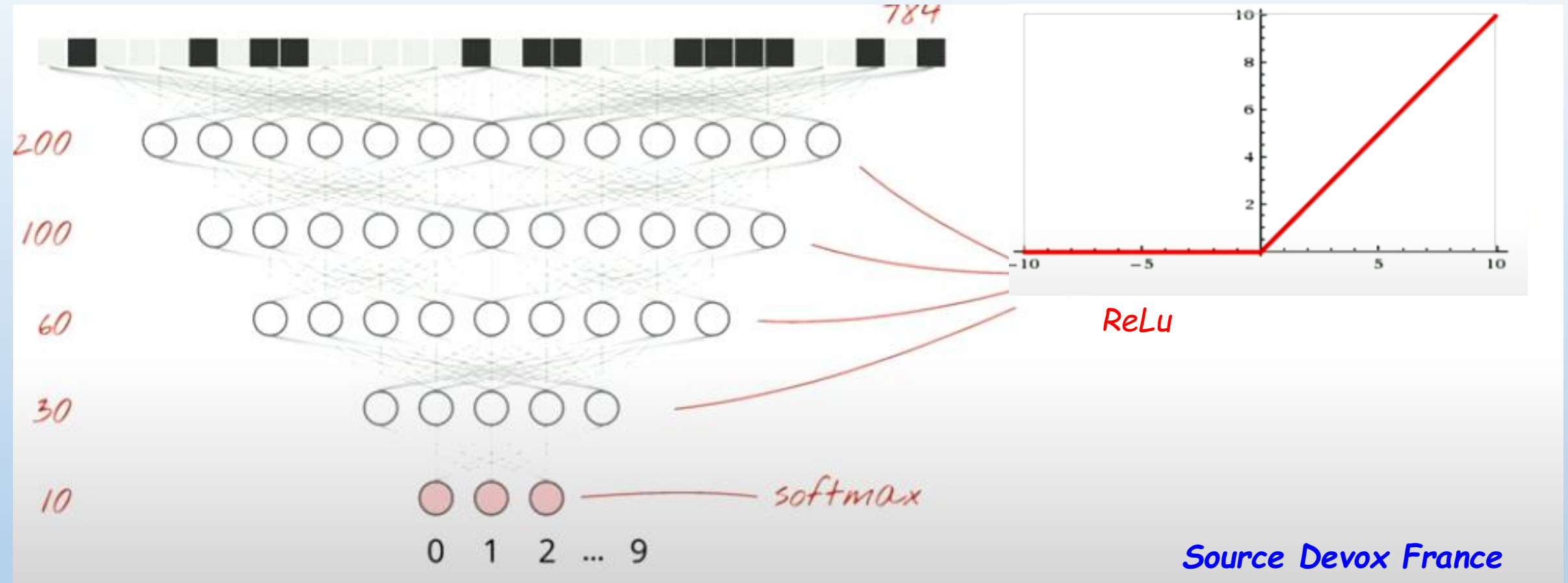
Google Developer relations

[@martin_gorner](#)

plus.google.com/+MartinGorner

Deep learning

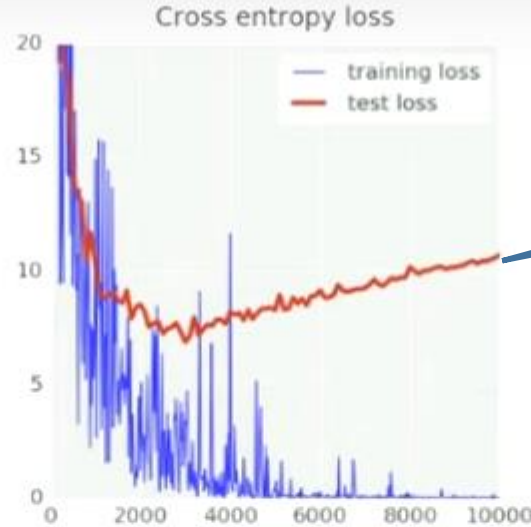
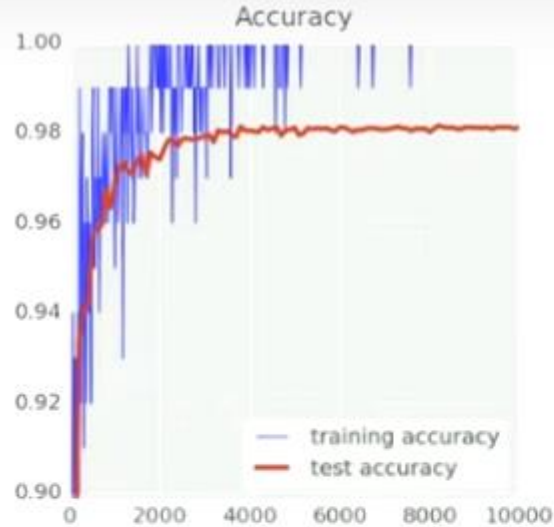
Modèle à 5 couches entièrement connectées FC



Modèle à 5 couches

Source Devox France

Perfo du modèle 100 % sur entrainement

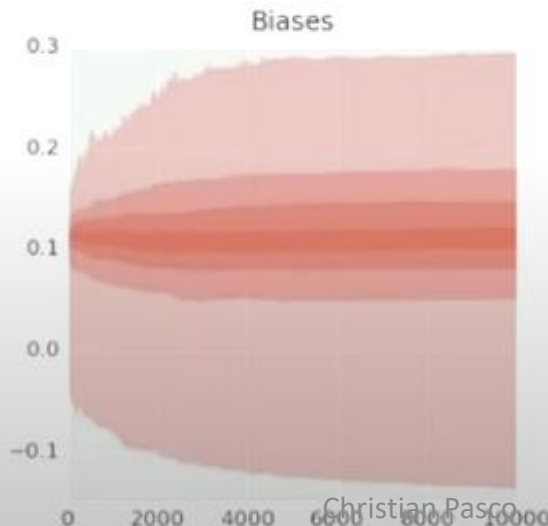
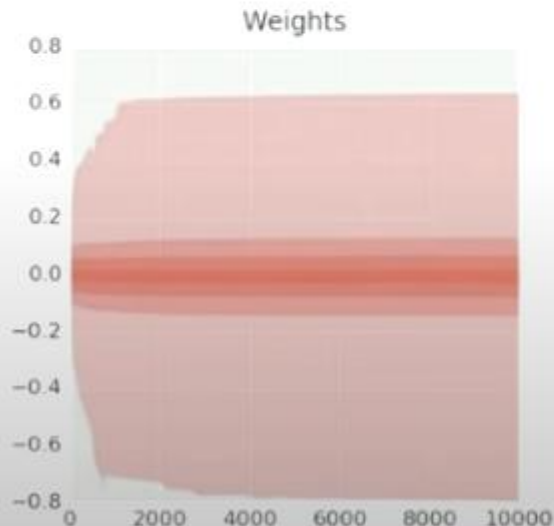


Training digits

5	0	0	2	4	6	7	0	3	9
4	0	1	7	5	6	9	6	0	6
0	6	1	0	0	6	0	6	3	9
8	1	7	4	6	3	6	9	5	9
0	5	1	6	3	2	7	5	7	3
4	7	0	8	1	5	5	2	7	1
6	4	4	2	1	3	6	7	3	4
0	7	8	0	7	0	1	1	7	4
3	3	7	0	3	8	0	1	8	9
4	4	7	9	5	0	8	1	6	1

Overfitting

Trop de neurones



98%

Perfo stagne

Difficultés

Overfitting

Performances

Lenteur apprentissage

Sensibilité à la translation

Comment améliorer ?

Réflexion :

Dans les années 80 => **Systemes Experts**

Règles :

- Si les pixels noirs ont la **forme** d'une seule boucle, alors c'est un 0
- si deux **boucles entrelacées** alors c'est un 8
- si **ligne verticale** au milieu , alors c'est un 1

Or, dans nos modèles nous prenons en entrée les valeurs des pixels sans tenir compte des caractéristiques, des formes etc.

Comment reconnaître et utiliser ces informations ?

Les réseaux convolutifs